Imitation-Projected Programmatic Reinforcement Learning

Abhinav Verma* Rice University averma@rice.edu
Hoang M. Le* Caltech hmle@caltech.edu
Yisong Yue Caltech yyue@caltech.edu
Swarat Chaudhuri Rice University swarat@rice.edu

Abstract

We study the problem of programmatic reinforcement learning, in which policies are represented as short programs in a symbolic language. Programmatic policies can be more interpretable, generalizable, and amenable to formal verification than neural policies; however, designing rigorous learning approaches for such policies remains a challenge. Our approach to this challenge — a meta-algorithm called PROPEL— is based on three insights. First, we view our learning task as optimization in policy space, modulo the constraint that the desired policy has a programmatic representation, and solve this optimization problem using a form of mirror descent that takes a gradient step into the unconstrained policy space and then projects back onto the constrained space. Second, we view the unconstrained policy space as mixing neural and programmatic representations, which enables employing state-of-the-art deep policy gradient approaches. Third, we cast the projection step as program synthesis via imitation learning, and exploit contemporary combinatorial methods for this task. We present theoretical convergence results for PROPEL and empirically evaluate the approach in three continuous control domains. The experiments show that PROPEL can significantly outperform state-of-the-art approaches for learning programmatic policies.

1 Introduction

A growing body of work [58, 8, 60] investigates reinforcement learning (RL) approaches that represent policies as programs in a symbolic language, e.g., a domain-specific language for composing control modules such as PID controllers [5]. Short programmatic policies offer many advantages over neural policies discovered through deep RL, including greater interpretability, better generalization to unseen environments, and greater amenability to formal verification. These benefits motivate developing effective approaches for learning such programmatic policies.

However, programmatic reinforcement learning (PRL) remains a challenging problem, owing to the highly structured nature of the policy space. Recent state-of-the-art approaches employ program synthesis methods to imitate or distill a pre-trained neural policy into short programs [58, 8]. However, such a distillation process can yield a highly suboptimal programmatic policy — i.e., a large distillation gap — and the issue of direct policy search for programmatic policies also remains open.

In this paper, we develop PROPEL (Imitation-**Pro**jected **P**rogrammatic R**e**inforcement **L**earning), a new learning meta-algorithm for PRL, as a response to this challenge. The design of PROPEL is based on three insights that enables integrating and building upon state-of-the-art approaches for policy gradients and program synthesis. First, we view programmatic policy learning as a constrained policy optimization problem, in which the desired policies are constrained to be those that have a programmatic representation. This insight motivates utilizing constrained mirror descent approaches, which take a gradient step into the unconstrained policy space and then project back onto the constrained space. Second, by allowing the unconstrained policy space to have a mix of neural

*Equal contribution

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

$$\begin{aligned}\pi(s) &::= a \mid Op(\pi_1(s), \ldots, \pi_k(s)) \mid \textbf{if } b \textbf{ then } \pi_1(s) \textbf{ else } \pi_2(s) \mid \oplus_\theta(\pi_1(s), \ldots, \pi_k(s)) \\ b &::= \phi(s) \mid BOp(b_1, \ldots, b_k)\end{aligned}$$

Figure 1: *A high-level syntax for programmatic policies, inspired by [58]. A policy $\pi(s)$ takes a state s as input and produces an action a as output. b represents boolean expressions; ϕ is a boolean-valued operator on states; Op is an operator that combines multiple policies into one policy; BOp is a standard boolean operator; and $\oplus_\theta$ is a "library function" parameterized by θ.*

$$\begin{aligned}&\textbf{if } (s[\texttt{TrackPos}] < 0.011 \textbf{ and } s[\texttt{TrackPos}] > -0.011) \\ &\qquad \textbf{then } \texttt{PID}_{\langle \texttt{RPM}, 0.45, 3.54, 0.03, 53.39 \rangle}(s) \textbf{ else } \texttt{PID}_{\langle \texttt{RPM}, 0.39, 3.54, 0.03, 53.39 \rangle}(s)\end{aligned}$$

Figure 2: *A programmatic policy for acceleration in* TORCS *[59], automatically discovered by* PROPEL. $s[\texttt{TrackPos}]$ *represents the most recent reading from sensor* `TrackPos`.

and programmatic representations, we can employ well-developed deep policy gradient approaches [55, 36, 47, 48, 19] to compute the unconstrained gradient step. Third, we define the projection operator using program synthesis via imitation learning [58, 8], in order to recover a programmatic policy from the unconstrained policy space. Our contributions can be summarized as:

- We present PROPEL, a novel meta-algorithm that is based on mirror descent, program synthesis, and imitation learning, for PRL.
- On the theoretical side, we show how to cast PROPEL as a form of constrained mirror descent. We provide a thorough theoretical analysis characterizing the impact of approximate gradients and projections. Further, we prove results that provide expected regret bounds and finite-sample guarantees under reasonable assumptions.
- On the practical side, we provide a concrete instantiation of PROPEL and evaluate it in three continuous control domains, including the challenging car-racing domain TORCS [59]. The experiments show significant improvements over state-of-the-art approaches for learning programmatic policies.

2 Problem Statement

The problem of programmatic reinforcement learning (PRL) consists of a Markov Decision Process (MDP) $\mathcal{M}$ and a programmatic policy class Π. The definition of $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, c, p_0, \gamma)$ is standard [54], with $\mathcal{S}$ being the state space, $\mathcal{A}$ the action space, $P(s'|s, a)$ the probability density function of transitioning from a state-action pair to a new state, $c(s, a)$ the state-action cost function, $p_0(s)$ a distribution over starting states, and $\gamma \in (0, 1)$ the discount factor. A policy $\pi : \mathcal{S} \to \mathcal{A}$ (stochastically) maps states to actions. We focus on continuous control problems, so $\mathcal{S}$ and $\mathcal{A}$ are assumed to be continuous spaces. The goal is to find a programmatic policy $\pi^* \in \Pi$ such that:

$$\pi^* = \operatorname*{argmin}_{\pi \in \Pi} J(\pi), \qquad \text{where: } J(\pi) = \mathbf{E}\left[\sum_{i=0}^{\infty} \gamma^i c(s_i, a_i \equiv \pi(s_i))\right], \tag{1}$$

with the expectation taken over the initial state distribution $s_0 \sim p_0$, the policy decisions, and the transition dynamics P. One can also use rewards, in which case (1) becomes a maximization problem.

Programmatic Policy Class. A programmatic policy class Π consists of policies that can be represented parsimoniously by a (domain-specific) programming language. Recent work [58, 8, 60] indicates that such policies can be easier to interpret and formally verify than neural policies, and can also be more robust to changes in the environment.

In this paper, we consider two concrete classes of programmatic policies. The first, a simplification of the class considered in Verma et al. [58], is defined by the modular, high-level language in Figure 1. This language assumes a library of parameterized functions $\oplus_\theta$ representing standard controllers, for instance Proportional-Integral-Derivative (PID) [6] or bang-bang controllers [11]. Programs in the language take states s as inputs and produce actions a as output, and can invoke fully instantiated library controllers along with predefined arithmetic, boolean and relational operators. The second, "lower-level" class, from Bastani et al. [8], consists of decision trees that map states to actions.

Example. Consider the problem of learning a programmatic policy, in the language of Figure 1, that controls a car's accelerator in the TORCS car-racing environment [59]. Figure 2 shows a program in our language for this task. The program invokes PID controllers $\texttt{PID}_{\langle j, \theta_P, \theta_I, \theta_D \rangle}$, where j identifies

Algorithm 1 Imitation-Projected Programmatic Reinforcement Learning (PROPEL)

Input: Programmatic & Neural Policy Classes: Π & $\mathcal{F}$.
Input: Either initial π_0 or initial f_0
Define joint policy class: $\mathcal{H} \equiv \Pi \oplus \mathcal{F}$ *//$h \equiv \pi + f$ defined as $h(s) = \pi(s) + f(s)$*
if given initial f_0 **then**
  $\pi_0 \leftarrow \text{PROJECT}(f_0)$ *//program synthesis via imitation learning*
end if
for $t = 1, \ldots, T$ **do**
  $h_t \leftarrow \text{UPDATE}_{\mathcal{F}}(\pi_{t-1}, \eta)$ *//policy gradient in neural policy space with learning rate η*
  $\pi_t \leftarrow \text{PROJECT}_{\Pi}(h_t)$ *//program synthesis via imitation learning*
end for
Return: Policy π_T

the sensor (out of 29, in our experiments) that provides inputs to the controller, and θ_P, θ_I, and θ_D are respectively the real-valued coefficients of the proportional, integral, and derivative terms in the controller. We note that the program only uses the sensors `TrackPos` and `RPM`. While `TrackPos` (for the position of the car relative to the track axis) is used to decide which controller to use, only the `RPM` sensor is needed to calculate the acceleration.

Learning Challenges. Learning programmatic policies in the continuous RL setting is challenging, as the best performing methods utilize policy gradient approaches [55, 36, 47, 48, 19], but policy gradients are hard to compute in programmatic representations. In many cases, Π may not even be differentiable. For our approach, we only assume access to program synthesis methods that can select a programmatic policy $\pi \in \Pi$ that minimizes imitation disagreement with demonstrations provided by a teaching oracle. Because imitation learning tends to be easier than general RL in long-horizon tasks [53], the task of imitating a neural policy with a program is, intuitively, significantly simpler than the full programmatic RL problem. This intuition is corroborated by past work on programmatic RL [58], which shows that direct search over programs often fails to meet basic performance objectives.

3 Learning Algorithm

To develop our approach, we take the viewpoint of (1) being a constrained optimization problem, where $\Pi \subset \mathcal{H}$ resides within a larger space of policies $\mathcal{H}$. In particular, we will represent $\mathcal{H} \equiv \Pi \oplus \mathcal{F}$ using a mixing of programmatic policies Π and neural polices $\mathcal{F}$. Any mixed policy $h \equiv \pi + f$ can be invoked as $h(s) = \pi(s) + f(s)$. In general, we assume that $\mathcal{F}$ is a good approximation of Π (i.e., for each $\pi \in \Pi$ there is some $f \in \mathcal{F}$ that approximates it well), which we formalize in Section 4.

We can now frame our constrained learning problem as minimizing (1) over $\Pi \subset \mathcal{H}$, that alternate between taking a gradient step in the general space $\mathcal{H}$ and projecting back down onto Π. This "lift-and-project" perspective motivates viewing our problem via the lens of mirror descent [40]. In standard mirror descent, the unconstrained gradient step can be written as $h \leftarrow h_{prev} - \eta \nabla_{\mathcal{H}} J(h_{prev})$ for step size η, and the projection can be written as $\pi \leftarrow \text{argmin}_{\pi' \in \Pi} D(\pi', h)$ for divergence measure D.

Our approach, *Imitation-Projected Programmatic Reinforcement Learning* (PROPEL), is outlined in Algorithm 1 (also see Figure 3). PROPEL is a meta-algorithm that requires instantiating two subroutines, UPDATE and PROJECT, which correspond to the standard update and projection steps, respectively. PROPEL can be viewed as a form of functional mirror descent with some notable deviations from vanilla mirror descent.

UPDATE$_{\mathcal{F}}$. Since policy gradient methods are well-developed for neural policy classes $\mathcal{F}$ (e.g., [36, 47, 48, 30, 24, 19]) and non-existent for programmatic policy classes Π, PROPEL is designed to leverage policy gradients in $\mathcal{F}$ and avoid policy gradients in Π. Algorithm 2 shows one instantiation of UPDATE$_{\mathcal{F}}$. Note that standard mirror descent takes unconstrained gradient steps in $\mathcal{H}$ rather than $\mathcal{F}$, and we discuss this discrepancy between UPDATE$_{\mathcal{H}}$ and UPDATE$_{\mathcal{F}}$ in Section 4.

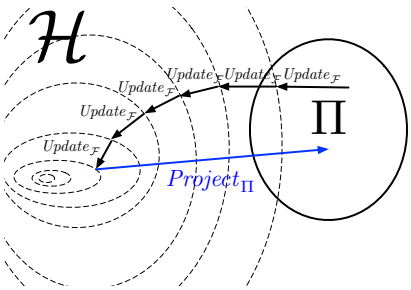

Figure 3: Depicting the PROPEL meta-algorithm.

PROJECT$_{\Pi}$. Projecting onto Π can be implemented using program synthesis via imitation learning, i.e., by synthesizing a $\pi \in \Pi$ to best imitate demonstrations provided by a teaching oracle $h \in \mathcal{H}$. Recent work [58, 8, 60] has given practical heuristics for this task for various

Algorithm 2 UPDATE$_{\mathcal{F}}$: neural policy gradient for mixed policies

Input: Neural Policy Class $\mathcal{F}$. **Input:** Reference programmatic policy: π
Input: Step size: η. **Input:** Regularization parameter: λ
Initialize neural policy: f_0 *//any standard randomized initialization*
for $j = 1, \ldots, m$ **do**
  $f_j \leftarrow f_{j-1} - \eta\lambda\nabla_{\mathcal{F}} J(\pi + \lambda f_{j-1})$ *//using DDPG [36], TRPO [47], etc., holding π fixed*
end for
Return: $h \equiv \pi + \lambda f_m$

Algorithm 3 PROJECT$_{\Pi}$: program synthesis via imitation learning

Input: Programmatic Policy Class: Π. **Input:** Oracle policy: h
Roll-out h on environment, get trajectory: $\tau_0 = (s^0, h(s^0), s^1, h(s^1), \ldots)$
Create supervised demonstration set: $\Gamma_0 = \{(s, h(s))\}$ from τ_0
Derive π_0 from Γ_0 via program synthesis *//e.g., using methods in [58, 8]*
for $k = 1, \ldots, M$ **do**
  Roll-out π_{k-1}, creating trajectory: τ_k
  Collect demonstration data: $\Gamma' = \{(s, h(s)) | s \in \tau_k\}$
  $\Gamma_k \leftarrow \Gamma' \cup \Gamma_{k-1}$ *//DAgger-style imitation learning [46]*
  Derive π_k from Γ_k via program synthesis *//e.g., using methods in [58, 8]*
end for
Return: π_M

programmatic policy classes. Algorithm 3 shows one instantiation of PROJECT$_{\Pi}$ (based on DAgger [46]). One complication that arises is that finite-sample runs of such imitation learning approaches only return approximate solutions and so the projection is not exact. We characterize the impact of approximate projections in Section 4.

Practical Considerations. In practice, we often employ multiple gradient steps before taking a projection step (as also described in Algorithm 2), because the step size of individual (stochastic) gradient updates can be quite small. Another issue that arises in virtually all policy gradient approaches is that the gradient estimates can have very high variance [55, 33, 30]. We utilize low-variance policy gradient updates by using the reference π as a proximal regularizer in function space [19].

For the projection step (Algorithm 3), in practice we often retain all previous roll-outs τ from all previous projection steps. It is straightforward to query the current oracle h to provide demonstrations on the states $s \in \tau$ from previous roll-outs, which can lead to substantial savings in sample complexity with regards to executing roll-outs on the environment, while not harming convergence.

4 Theoretical Analysis

We start by viewing PROPEL through the lens of online learning in function space, independent of the specific parametric representation. This start point yields a convergence analysis of Alg. 1 in Section 4.1 under generic approximation errors. We then analyze the issues of policy class representation in Sections 4.2 and 4.3, and connect Algorithms 2 and 3 with the overall performance, under some simplifying conditions. In particular, Section 4.3 characterizes the update error in a possibly non-differentiable setting; to our knowledge, this is the first such analysis of its kind for reinforcement learning.

Preliminaries. We consider Π and $\mathcal{F}$ to be subspaces of an ambient policy space $\mathcal{U}$, which is a vector space equipped with inner product $\langle \cdot, \cdot \rangle$, induced norm $\|u\| = \sqrt{\langle u, u \rangle}$, dual norm $\|v\|_* = \sup\{\langle v, u \rangle | \, \|u\| \leq 1\}$, and standard scaling & addition: $(au + bv)(s) = au(s) + bv(s)$ for $a, b \in \mathbb{R}$ and $u, v \in \mathcal{U}$. The cost functional of a policy u is $J(u) = \int_{\mathcal{S}} c(s, u(s)) d\mu^u(s)$, where μ^u is the distribution of states induced by u. The joint policy class is $\mathcal{H} = \Pi \oplus \mathcal{F}$, by $\mathcal{H} = \{\pi + f | \forall \pi \in \Pi, f \in \mathcal{F}\}$.[2] Note that $\mathcal{H}$ is a subspace of $\mathcal{U}$, and inherits its vector space properties. Without affecting the analysis, we simply equate $\mathcal{U} \equiv \mathcal{H}$ for the remainder of the paper.

We assume that J is convex in $\mathcal{H}$, which implies that subgradient $\partial J(h)$ exists (with respect to $\mathcal{H}$) [9]. Where J is differentiable, we utilize the notion of a Fréchet gradient. Recall that a bounded linear operator $\nabla : \mathcal{H} \mapsto \mathcal{H}$ is called a Fréchet functional gradient of J at $h \in \mathcal{H}$ if

[2]The operator $\oplus$ is not a direct sum, since Π and $\mathcal{F}$ are not orthogonal.

$\lim_{\|g\|\to 0} \frac{J(h+g)-J(h)-\langle \nabla J(h), g\rangle}{\|g\|} = 0$. By default, ∇ (or $\nabla_{\mathcal{H}}$ for emphasis) denotes the gradient with respect to $\mathcal{H}$, whereas $\nabla_{\mathcal{F}}$ defines the gradient in the restricted subspace $\mathcal{F}$.

4.1 PROPEL as (Approximate) Functional Mirror Descent

For our analysis, PROPEL can be viewed as approximating mirror descent in (infinite-dimensional) function space over a convex set $\Pi \subset \mathcal{H}$.[3] Similar to the finite-dimensional setting [40], we choose a strongly convex and smooth functional regularizer R to be the mirror map. From the approximate mirror descent perspective, for each iteration t:

1. Obtain a noisy gradient estimate: $\widehat{\nabla}_{t-1} \approx \nabla J(\pi_{t-1})$
2. $\text{UPDATE}_{\mathcal{H}}(\pi)$ in $\mathcal{H}$ space: $\nabla R(h_t) = \nabla R(\pi_{t-1}) - \eta \widehat{\nabla}_{t-1}$ *(Note* $\text{UPDATE}_{\mathcal{H}} \neq \text{UPDATE}_{\mathcal{F}}$*)*
3. Obtain approximate projection: $\pi_t = \text{PROJECT}^R_{\Pi}(h_t) \approx \text{argmin}_{\pi\in\Pi} D_R(\pi, h_t)$

$D_R(u, v) = R(u) - R(v) - \langle \nabla R(u), u - v\rangle$ is a Bregman divergence. Taking $R(h) = \frac{1}{2}\|h\|^2$ will recover projected functional gradient descent in $L2$-space. Here UPDATE becomes $h_t = \pi_{t-1} - \eta \widehat{\nabla} J(\pi_{t-1})$, and PROJECT solves for $\text{argmin}_{\pi\in\Pi} \|\pi - h_t\|^2$. While we mainly focus on this choice of R in our experiments, note that other selections of R lead to different UPDATE and PROJECT operators (e.g., minimizing KL divergence if R is negative entropy).

The functional mirror descent scheme above may encounter two additional sources of error compared to standard mirror descent [40]. First, in the stochastic setting (also called bandit feedback [28]), the gradient estimate $\widehat{\nabla}_t$ may be biased, in addition to having high variance. One potential source of bias is the gap between $\text{UPDATE}_{\mathcal{H}}$ and $\text{UPDATE}_{\mathcal{F}}$. Second, the PROJECT step may be inexact. We start by analyzing the behavior of PROPEL under generic bias, variance, and projection errors, before discussing the implications of approximating $\text{UPDATE}_{\mathcal{H}}$ and PROJECT_{Π} by Algs. 2 & 3, respectively. Let the bias be bounded by β, i.e., $\left\|\mathbb{E}[\widehat{\nabla}_t|\pi_t] - \nabla J(\pi_t)\right\|_* \leq \beta$ almost surely. Similarly let the variance of the gradient estimate be bounded by σ^2, and the projection error norm $\|\pi_t - \pi_t^*\| \leq \epsilon$. We state the expected regret bound below; more details and a proof appear in Appendix A.2.

Theorem 4.1 (Expected regret bound under gradient estimation and projection errors)**.** *Let $\pi_1, \ldots, \pi_T$ be a sequence of programmatic policies returned by Algorithm 1, and π^* be the optimal programmatic policy. Choosing learning rate $\eta = \sqrt{\frac{1}{\sigma^2}(\frac{1}{T} + \epsilon)}$, we have the expected regret over T iterations:*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\sigma\sqrt{\frac{1}{T} + \epsilon} + \beta\right). \tag{2}$$

The result shows that error ϵ from PROJECT and the bias β do not accumulate and simply contribute an additive term on the expected regret.[4] The effect of variance of gradient estimate decreases at a $\sqrt{1/T}$ rate. Note that this regret bound is agnostic to the specific UPDATE and PROJECT operations, and can be applied more generically beyond the specific algorithmic choices used in our paper.

4.2 Finite-Sample Analysis under Vanilla Policy Gradient Update and DAgger Projection

Next, we show how certain instantiations of UPDATE and PROJECT affect the magnitude of errors and influence end-to-end learning performance from finite samples, under some simplifying assumptions on the UPDATE step. For this analysis, we simplify Alg. 2 into the case $\text{UPDATE}_{\mathcal{F}} \equiv \text{UPDATE}_{\mathcal{H}}$. In particular, we assume programmatic policies in Π to be parameterized by a vector $\theta \in \mathbb{R}^k$, and π is differentiable in θ (e.g., we can view $\Pi \subset \mathcal{F}$ where $\mathcal{F}$ is parameterized in $\mathbb{R}^k$). We further assume the trajectory roll-out is performed in an exploratory manner, where action is taken uniformly random over finite set of A actions, thus enabling the bound on the bias of gradient estimates via Bernstein's inequality. The PROJECT step is consistent with Alg. 3, i.e., using DAgger [45] under convex imitation loss, such as ℓ_2 loss. We have the following high-probability guarantee:

Theorem 4.2 (Finite-sample guarantee)**.** *At each iteration, we perform vanilla policy gradient estimate of π (over $\mathcal{H}$) using m trajectories and, use DAgger algorithm to collect M roll-outs for the*

[3] Π can be convexified by considering *randomized* policies, as stochastic combinations of $\pi \in \Pi$ (cf. [35]).

[4] Other mirror descent-style analyses, such as in [52], lead to accumulation of errors over the rounds of learning T. One key difference is that we are leveraging the assumption of convexity of J in the (infinite-dimensional) function space representation.

imitation learning projection. Setting the learning rate $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}\right)}$*, after* T *rounds of the algorithm, we have that:*

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leq O\left(\sigma\sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}\right) + O\left(\sigma\sqrt{\frac{\log(Tk/\delta)}{m}} + \frac{AH\log(Tk/\delta)}{m}\right)$$

holds with probability at least $1-\delta$*, with* H *being the task horizon,* A *the cardinality of action space,* σ^2 *the variance of policy gradient estimates, and* k *the dimension* Π*'s parameterization.*

The expanded result and proof are included in Appendix A.3. The proof leverages previous analysis from DAgger [46] and the finite sample analysis of vanilla policy gradient algorithm [32]. The finite-sample regret bound scales linearly with the standard deviation σ of the gradient estimate, while the bias, which is the very last component of the RHS, scales linearly with the task horizon H. Note that the standard deviation σ can be exponential in task horizon H in the worst case [32], and so it is important to have practical implementation strategies to reduce the variance of the UPDATE operation. While conducted in a stylized setting, this analysis provides insight in the relative trade-offs of spending effort in obtaining more accurate projections versus more reliable gradient estimates.

4.3 Closing the gap between UPDATE$_\mathcal{H}$ and UPDATE$_\mathcal{F}$

Our functional mirror descent analysis rests on taking gradients in $\mathcal{H}$: UPDATE$_\mathcal{H}(\pi)$ involves estimating $\nabla_\mathcal{H} J(\pi)$ in the $\mathcal{H}$ space. On the other hand, Algorithm 2 performs UPDATE$_\mathcal{F}(\pi)$ only in the neural policy space $\mathcal{F}$. In either case, although $J(\pi)$ may be differentiable in the non-parametric ambient policy space, it may not be possible to obtain a differentiable parametric programmatic representation in Π. In this section, we discuss theoretical motivations to addressing a practical issue: *How do we define and approximate the gradient* $\nabla_\mathcal{H} J(\pi)$ *under a parametric representation?* To our knowledge, we are the first to consider such a theoretical question for reinforcement learning.

Defining a consistent approximation of $\nabla_\mathcal{H} J(\pi)$**.** The idea in UPDATE$_\mathcal{F}(\pi)$ (Line 8 of Alg. 1) is to approximate $\nabla_\mathcal{H} J(\pi)$ by $\nabla_\mathcal{F} J(f)$, which has a differentiable representation, at some f close to π (under the norm). Under appropriate conditions on $\mathcal{F}$, we show that this approximation is valid.

Proposition 4.3. *Assume that (i)* J *is Fréchet differentiable on* $\mathcal{H}$*, (ii)* J *is also differentiable on the restricted subspace* $\mathcal{F}$*, and (iii)* $\mathcal{F}$ *is dense in* $\mathcal{H}$ *(i.e., the closure* $\overline{\mathcal{F}} = \mathcal{H}$*). Then for any fixed policy* $\pi \in \Pi$*, define a sequence of policies* $f_k \in \mathcal{F}$*,* $k = 1, 2, \ldots$*), that converges to* π*:* $\lim_{k\to\infty} \|f_k - \pi\| = 0$*. We then have* $\lim_{k\to\infty} \|\nabla_\mathcal{F} J(f_k) - \nabla_\mathcal{H} J(\pi)\|_* = 0$*.*

Since the Fréchet gradient is unique in the ambient space $\mathcal{H}$, $\forall k$ we have $\nabla_\mathcal{H} J(f_k) = \nabla_\mathcal{F} J(f_k) \to \nabla_\mathcal{H} J(\pi)$ as $k \to \infty$ (by Proposition 4.3). We thus have an asymptotically unbiased approximation of $\nabla_\mathcal{H} J(\pi)$ via differentiable space $\mathcal{F}$ as: $\nabla_\mathcal{F} J(\pi) \triangleq \nabla_\mathcal{H} J(\pi) \triangleq \lim_{k\to\infty} \nabla_\mathcal{F} J(f_k)$.[5] Connecting to the result from Theorem 4.1, let σ^2 be an upper bound on the policy gradient estimates in the *neural policy class* $\mathcal{F}$, under an asymptotically unbiased approximation of $\nabla_\mathcal{H} J(\pi)$, the expected regret bound becomes $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\sigma\sqrt{\frac{1}{T} + \epsilon}\right)$.

Bias-variance considerations of UPDATE$_\mathcal{F}(\pi)$ To further theoretically motivate a practical strategy for UPDATE$_\mathcal{F}(\pi)$ in Algorithm 2, we utilize an equivalent proximal perspective of mirror descent [10], where UPDATE$_\mathcal{H}(\pi)$ is equivalent to solving for $h' = \operatorname{argmin}_{h\in\mathcal{H}} \eta\langle\nabla_\mathcal{H} J(\pi), h\rangle + D_R(h, \pi)$.

Proposition 4.4 (Minimizing a relaxed objective)**.** *For a fixed programmatic policy* π*, with sufficiently small constant* $\lambda \in (0, 1)$*, we have that*

$$\min_{h\in\mathcal{H}} \eta\langle\nabla_\mathcal{H} J(\pi), h)\rangle + D_R(h, \pi) \leq \min_{f\in\mathcal{F}} J\big(\pi + \lambda f\big) - J(\pi) + \langle\nabla J(\pi), \pi\rangle \tag{3}$$

Thus, a relaxed UPDATE$_\mathcal{H}$ step is obtained by minimizing the RHS of (3), i.e., minimizing $J(\pi + \lambda f)$ over $f \in \mathcal{F}$. Each gradient descent update step is now $f' = f - \eta\lambda\nabla_\mathcal{F} J(\pi_t + \lambda f)$, corresponding to Line 5 of Algorithm 2. For fixed π and small λ, this relaxed optimization problem becomes regularized policy optimization over $\mathcal{F}$, which is significantly easier. Functional regularization in policy space around a fixed prior controller π has demonstrated significant reduction in the variance

[5] We do not assume $J(\pi)$ to be differentiable when restricting to the policy subspace Π, i.e., $\nabla_\Pi J(\pi)$ may not exist under policy parameterization of Π.

of gradient estimate [19], at the expense of some bias. The below expected regret bound summarizes the impact of this increased bias and reduced variance, with details included in Appendix A.5.

Proposition 4.5 (Bias-variance characterization of UPDATE$_{\mathcal{F}}$)**.** *Assuming $J(h)$ is L-strongly smooth over $\mathcal{H}$, i.e., $\nabla_{\mathcal{H}} J(h)$ is L-Lipschitz continuous, approximating* UPDATE$_{\mathcal{H}}$ *by* UPDATE$_F$ *per Alg. 2 leads to the expected regret bound:* $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\lambda\sigma\sqrt{\frac{1}{T}+\epsilon} + \lambda^2 L^2\right)$.

Compared to the idealized unbiased approximation in Proposition 4.3, the introduced bias here is related to the inherent smoothness property of cost functional $J(h)$ over the joint policy class $\mathcal{H}$, i.e., how close $J(\pi + \lambda f)$ is to its linear under-approximation $J(\pi) + \langle \nabla_{\mathcal{H}} J(\pi), \lambda f \rangle$ around π.

5 Experiments

We demonstrate the effectiveness of PROPEL in synthesizing programmatic controllers in three continuous control environments. For brevity and focus, this section primarily focuses on TORCS[6], a challenging race car simulator environment [59]. Empirical results on two additional classic control tasks, Mountain-Car and Pendulum, are provided in Appendix B; those results follow similar trends as the ones described for TORCS below, and further validate the convergence analysis of PROPEL.

Experimental Setup. We evaluate over five distinct tracks in the TORCS simulator. The difficulty of a track can be characterized by three properties; track length, track width, and number of turns. Our suite of tracks provides environments with varying levels of difficulty for the learning algorithm. The performance of a policy in the TORCS simulator is measured by the *lap time* achieved on the track. To calculate the lap time, the policies are allowed to complete a three-lap race, and we record the best lap time during this race. We perform the experiments with twenty-five random seeds and report the median lap time over these twenty-five trials. Some of the policies crash the car before completing a lap on certain tracks, even after training for 600 episodes. Such crashes are recorded as a lap time of infinity while calculating the median. If the policy crashes for more than half the seeds, this is reported as CR in Tables 1 & 2. We choose to report the median because taking the crash timing as infinity, or an arbitrarily large constant, heavily skews other common measures such as the mean.

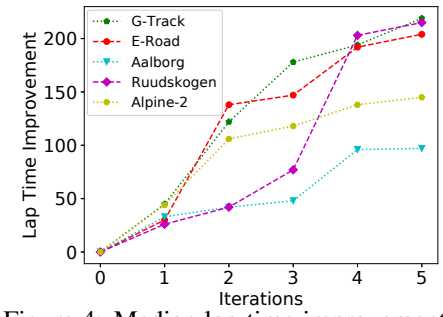

Figure 4: Median lap-time improvements during multiple iterations of PROPELPROG over 25 random seeds.

Baselines. Among recent state-of-the-art approaches to learning programmatic policies are NDPS [58] for high-level language policies, and VIPER [8] for learning tree-based policies. Both NDPS and VIPER rely on imitating a fixed (pre-trained) neural policy oracle, and can be viewed as degenerate versions of PROPEL that only run Lines 4-6 in Algorithm 1. We present two PROPEL analogues to NDPS and VIPER: (i) PROPELPROG: PROPEL using the high-level language of Figure 1 as the class of programmatic policies, similar to NDPS. (ii) PROPELTREE: PROPEL using regression trees, similar to VIPER. We also report results for PRIOR, which is a (sub-optimal) PID controller that is also used as the initial policy in PROPEL. In addition, to study generalization ability as well as safety behavior during training, we also include DDPG, a neural policy learned using the Deep Deterministic Policy Gradients [36] algorithm, with 600 episodes of training for each track. In principle, PROPEL and its analysis can accommodate different policy gradient subroutines. However, in the TORCS domain, other policy gradient algorithms such as PPO and TRPO failed to learn policies that are able to complete the considered tracks. We thus focus on DDPG as our main policy gradient component.

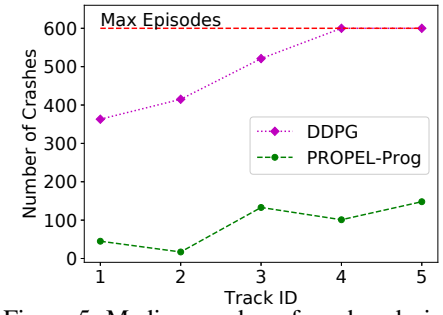

Figure 5: Median number of crashes during training of DDPG and PROPELPROG over 25 random seeds.

[6]The code for the TORCS experiments can be found at: https://bitbucket.org/averma8053/propel

Table 1: *Performance results in* TORCS *over 25 random seeds. Each entry is formatted as Lap-time / Crash-ratio, reporting median lap time in seconds over all the seeds (lower is better) and ratio of seeds that result in crashes (lower is better). A lap time of* CR *indicates the agent crashed and could not complete a lap for more than half the seeds.*

	G-TRACK	E-ROAD	AALBORG	RUUDSKOGEN	ALPINE-2
LENGTH	3186M	3260M	2588M	3274M	3774M
PRIOR	312.92 / 0.0	322.59 / 0.0	244.19 / 0.0	340.29 / 0.0	402.89 / 0.0
DDPG	78.82 / 0.24	89.71 / 0.28	101.06 / 0.40	CR / 0.68	CR / 0.92
NDPS	108.25 / 0.24	126.80 / 0.28	163.25 / 0.40	CR / 0.68	CR / 0.92
VIPER	83.60 / 0.24	87.53 / 0.28	110.57 / 0.40	CR / 0.68	CR / 0.92
PROPELPROG	93.67 / 0.04	119.17 / 0.04	147.28 / 0.12	124.58 / 0.16	256.59 / 0.16
PROPELTREE	78.33 / 0.04	79.39 / 0.04	109.83 / 0.16	118.80 / 0.24	236.01 / 0.36

Table 2: *Generalization results in* TORCS*, where rows are training and columns are testing tracks. Each entry is formatted as* PROPELPROG */ DDPG, and the number reported is the median lap time in seconds over all the seeds (lower is better).* CR *indicates the agent crashed and could not complete a lap for more than half the seeds.*

	G-TRACK	E-ROAD	AALBORG	RUUDSKOGEN	ALPINE-2
G-TRACK	-	124 / CR	CR / CR	CR / CR	CR / CR
E-ROAD	102 / 92	-	CR / CR	CR / CR	CR / CR
AALBORG	201 / 91	228 / CR	-	217 / CR	CR / CR
RUUDSKOGEN	131 / CR	135 / CR	CR / CR	-	CR / CR
ALPINE-2	222 / CR	231 / CR	184 / CR	CR / CR	-

Evaluating Performance. Table 1 shows the performance on the considered TORCS tracks. We see that PROPELPROG and PROPELTREE consistently outperform the NDPS [58] and VIPER [8] baselines, respectively. While DDPG outperforms PROPEL on some tracks, its volatility causes it to be unable to learn in some environments, and hence to crash the majority of the time. Figure 4 shows the consistent improvements made over the prior by PROPELPROG, over the iterations of the PROPEL algorithm. Appendix B contains similar results achieved on the two classic control tasks, MountainCar and Pendulum. Figure 5 shows that, compared to DDPG, our approach suffers far fewer crashes while training in TORCS.

Evaluating Generalization. To compare the ability of the controllers to perform on tracks not seen during training, we executed the learned policies on all the other tracks (Table 2). We observe that DDPG crashes significantly more often than PROPELPROG. This demonstrates the generalizability of the policies returned by PROPEL. Generalization results for the PROPELTREE policy are given in the appendix. In general, PROPELTREE policies are more generalizable than DDPG but less than PROPELPROG. On an absolute level, the generalization ability of PROPEL still leaves much room for improvement, which is an interesting direction for future work.

Verifiability of Policies. As shown in prior work [8, 58], parsimonious programmatic policies are more amenable to formal verification than neural policies. Unsurprisingly, the policies generated by PROPELTREE and PROPELPROG are easier to verify than DDPG policies. As a concrete example, we verified a smoothness property of the PROPELPROG policy using the Z3 SMT-solver [21] (more details in Appendix B). The verification terminated in 0.49 seconds.

Initialization. In principle, PROPEL can be initialized with a random program, or a random policy trained using DDPG. In practice, the performance of PROPEL depends to a certain degree on the stability of the policy gradient procedure, which is DDPG in our experiments. Unfortunately, DDPG often exhibits high variance across trials and fares poorly in challenging RL domains. Specifically, in our TORCS experiments, DDPG fails on a number of tracks (similar phenomena have been reported in previous work that experiments on similar continuous control domains [30, 19, 58]). Agents obtained by initializing PROPEL with neural policies obtained via DDPG also fail on multiple tracks. Their performance over the five tracks is reported in Appendix B. In contrast, PROPEL can often finish the challenging tracks when initialized with a very simple hand-crafted programmatic prior.

6 Related Work

Program Synthesis. Program synthesis is the problem of automatically searching for a program within a language that fits a given specification [29]. Recent approaches to the problem have leveraged symbolic knowledge about program structure [27], satisfiability solvers [50, 31], and meta-learning techniques [39, 41, 22, 7] to generate interesting programs in many domains [3, 42, 4]. In most prior work, the specification is a logical constraint on the input/output behavior of the target program. However, there is also a growing body of work that considers program synthesis modulo optimality objectives [13, 15, 43], often motivated by machine learning tasks [39, 57, 26, 23, 58, 8, 60]. Synthesis of programs that imitates an oracle has been considered in both the logical [31] and the optimization [58, 8, 60] settings. The projection step in PROPEL builds on this prior work. While our current implementation of this step is entirely symbolic, in principle, the operation can also utilize contemporary techniques for learning policies that guide the synthesis process [39, 7, 49].

Constrained Policy Learning. Constrained policy learning has seen increased interest in recent years, largely due to the desire to impose side guarantees such as stability and safety on the policy's behavior. Broadly, there are two approaches to imposing constraints: specifying constraints as an additional cost function [1, 35], and explicitly encoding constraints into the policy class [2, 34, 19, 20, 12]. In some cases, these two approaches can be viewed as duals of each other. For instance, recent work that uses control-theoretic policies as a functional regularizer [34, 19] can be viewed from the perspective of both regularization (additional cost) and an explicitly constrained policy class (a specific mix of neural and control-theoretic policies). We build upon this perspective to develop the gradient update step in our approach.

RL using Imitation Learning. There are two ways to utilize imitation learning subroutines within RL. First, one can leverage limited-access or sub-optimal experts to speed up learning [44, 18, 14, 51]. Second, one can learn over two policy classes (or one policy and one model class) to achieve accelerated learning compared to using only one policy class [38, 17, 52, 16]. Our approach has some stylistic similarities to previous efforts [38, 52] that use a richer policy space to search for improvements before re-training the primary policy to imitate the richer policy. One key difference is that our primary policy is programmatic and potentially non-differentiable. A second key difference is that our theoretical framework takes a functional gradient descent perspective — it would be interesting to carefully compare with previous analysis techniques to find a unifying framework.

RL with Mirror Descent. The mirror descent framework has previously used to analyze and design RL algorithms. For example, Thomas et al. [56] and Mahadevan and Liu [37] use composite objective mirror descent, or COMID [25], which allows incorporating adaptive regularizers into gradient updates, thus offering connections to either natural gradient RL [56] or sparsity inducing RL algorithms [37]. Unlike in our work, these prior approaches perform projection into the same native, differentiable representation. Also, the analyses in these papers do not consider errors introduced by hybrid representations and approximate projection operators. However, one can potentially extend our approach with versions of mirror descent, e.g., COMID, that were considered in these efforts.

7 Conclusion and Future Work

We have presented PROPEL, a meta-algorithm based on mirror descent, program synthesis, and imitation learning, for programmatic reinforcement learning (PRL). We have presented theoretical convergence results for PROPEL, developing novel analyses to characterize approximate projections and biased gradients within the mirror descent framework. We also validated PROPEL empirically, and show that it can discover interpretable, verifiable, generalizable, performant policies and significantly outperform the state of the art in PRL.

The central idea of PROPEL is the use of imitation learning and combinatorial methods in implementing a projection operation for mirror descent, with the goal of optimization in a functional space that lacks gradients. While we have developed PROPEL in an RL setting, this idea is not restricted to RL or even sequential decision making. Future work will seek to exploit this insight in other machine learning and program synthesis settings.

Acknowledgements. This work was supported in part by United States Air Force Contract # FA8750-19-C-0092, NSF Award # 1645832, NSF Award # CCF-1704883, the Okawa Foundation, Raytheon, PIMCO, and Intel.

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org, 2017.

[2] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, pages 1–25. 2015.

[4] Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. Scaling enumerative program synthesis via divide and conquer. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, pages 319–336, 2017.

[5] Kiam Heong Ang, Gregory Chong, and Yun Li. Pid control system analysis, design, and technology. *IEEE transactions on control systems technology*, 13(4):559–576, 2005.

[6] Karl Johan Åström and Tore Hägglund. Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica*, 20(5):645–651, 1984.

[7] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[8] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2494–2504, 2018.

[9] Heinz H Bauschke, Patrick L Combettes, et al. *Convex analysis and monotone operator theory in Hilbert spaces*, volume 408. Springer, 2011.

[10] Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

[11] Richard Bellman, Irving Glicksberg, and Oliver Gross. On the "bang-bang" control problem. *Quarterly of Applied Mathematics*, 14(1):11–18, 1956.

[12] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[13] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pages 140–156, 2009.

[14] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. In *International Conference on Machine Learning (ICML)*, 2015.

[15] Swarat Chaudhuri, Martin Clochard, and Armando Solar-Lezama. Bridging boolean and quantitative synthesis using smoothed proof search. In *POPL*, pages 207–220, 2014.

[16] Ching-An Cheng, Xinyan Yan, Nathan Ratliff, and Byron Boots. Predictor-corrector policy optimization. In *International Conference on Machine Learning (ICML)*, 2019.

[17] Ching-An Cheng, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Accelerating imitation learning with predictive models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

[18] Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. Fast policy learning through imitation and reinforcement. In *Uncertainty in artificial intelligence*, 2019.

[19] Richard Cheng, Abhinav Verma, Gabor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel Burdick. Control regularization for reduced variance reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2019.

[20] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[21] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, pages 337–340, 2008.

[22] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 990–998. JMLR. org, 2017.

[23] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: automatic conversion of 3d models to CSG trees. *ACM Trans. Graph.*, 37(6):213:1–213:16, 2018.

[24] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[25] John C Duchi, Shai Shalev-Shwartz, Yoram Singer, and Ambuj Tewari. Composite objective mirror descent. In *COLT*, pages 14–26, 2010.

[26] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems*, pages 6059–6068, 2018.

[27] John K. Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 229–239, 2015.

[28] Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 385–394. Society for Industrial and Applied Mathematics, 2005.

[29] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1-2):1–119, 2017.

[30] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[31] Susmit Jha, Sumit Gulwani, Sanjit A Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 215–224. ACM, 2010.

[32] Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

[33] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[34] Hoang M. Le, Andrew Kang, Yisong Yue, and Peter Carr. Smooth imitation learning for online sequence prediction. In *International Conference on Machine Learning (ICML)*, 2016.

[35] Hoang M Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning (ICML)*, 2019.

[36] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[37] Sridhar Mahadevan and Bo Liu. Sparse q-learning with mirror descent. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 564–573. AUAI Press, 2012.

[38] William H Montgomery and Sergey Levine. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016, 2016.

[39] Vijayaraghavan Murali, Swarat Chaudhuri, and Chris Jermaine. Neural sketch learning for conditional program generation. In *ICLR*, 2018.

[40] Arkadii Semenovich Nemirovsky and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.

[41] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.

[42] Oleksandr Polozov and Sumit Gulwani. Flashmeta: a framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, pages 107–126, 2015.

[43] Veselin Raychev, Pavol Bielik, Martin T. Vechev, and Andreas Krause. Learning programs from noisy data. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 761–774, 2016.

[44] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[45] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[46] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 627–635, 2011.

[47] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[48] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[49] Xujie Si, Yuan Yang, Hanjun Dai, Mayur Naik, and Le Song. Learning a meta-solver for syntax-guided program synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[50] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415, 2006.

[51] Wen Sun, J Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[52] Wen Sun, Geoffrey J Gordon, Byron Boots, and J Bagnell. Dual policy iteration. In *Advances in Neural Information Processing Systems*, pages 7059–7069, 2018.

[53] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International Conference on Machine Learning (ICML)*, 2017.

[54] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[55] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[56] Philip S Thomas, William C Dabney, Stephen Giguere, and Sridhar Mahadevan. Projected natural actor-critic. In *Advances in neural information processing systems*, pages 2337–2345, 2013.

[57] Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems*, pages 8687–8698, 2018.

[58] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5052–5061, 2018.

[59] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. TORCS, The Open Racing Car Simulator. http://www.torcs.org, 2014.

[60] He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *ACM Conference on Programming Language Design and Implementation (SIGPLAN)*, 2019.

A Theoretical Analysis

A.1 Preliminaries and Notations

We formally define an ambient control policy space $\mathcal{U}$ to be a vector space equipped with inner product $\langle \cdot, \cdot \rangle : \mathcal{U} \times \mathcal{U} \mapsto \mathbb{R}$, which induces a norm $\|u\| = \sqrt{\langle u, u \rangle}$, and its dual norm defined as $\|v\|_* = \sup\{\langle v, u \rangle | \, \|u\| \leq 1\}$. While multiple ways to define the inner product exist, for concreteness we can think of the example of square-integrable stationary policies with $\langle u, v \rangle = \int_{\mathcal{S}} u(s)v(s)ds$. The addition operator $+$ between two policies $u, v \in \mathcal{U}$ is defined as $(u + v)(s) = u(s) + v(s)$ for all state $s \in \mathcal{S}$. Scaling $\lambda u + \kappa v$ is defined similarly for scalar λ, κ.

The cost functional of a control policy u is defined as $J(u) = \int_0^\infty c(s(\tau), u(\tau))d\tau$, or $J(u) = \int_{\mathcal{S}} c(s, u(s))d\mu^u(s)$, where μ^u is the distribution of states induced by policy u. This latter example is equivalent to the standard notion of value function in reinforcement learning.

Separate from the parametric representation issues, both programmatic policy class Π and neural policy class $\mathcal{F}$, and by extension - the joint policy class $\mathcal{H}$, are considered to live in the ambient vector space $\mathcal{U}$. We thus have a common and well-defined notion of distance between policies from different classes.

We make an important distinction between differentiability of $J(h)$ in the ambient policy space (non-parametric), versus differentiability in parameterization (parametric). For example, if Π is a class of decision-tree based policy, policies in Π may not be differentiable under representation. However, policies $\pi \in \Pi$ might still be differentiable when considered as points in the ambient vector space $\mathcal{U}$.

We will use the following standard notion of gradient and differentiability from functional analysis:

Definition A.1 (Subgradients)**.** The subgradient of J at h, denoted $\partial J(h)$, is the non-empty set $\{g \in \mathcal{H} | \forall j \in \mathcal{H} : \langle j - h, g \rangle + J(h) \leq J(j)\}$

Definition A.2 (Fréchet gradient)**.** A bounded linear operator $\nabla : \mathcal{H} \mapsto \mathcal{H}$ is called Fréchet functional gradient of J at $h \in \mathcal{H}$ if $\lim_{\|g\| \to 0} \frac{J(h+g) - J(h) - \langle \nabla J(h), g \rangle}{\|g\|} = 0$

The notions of convexity, smoothness and Bregman divergence are analogous to finite-dimensional setting:

Definition A.3 (Strong convexity)**.** A differentiable function R is $\alpha-$strongly convex w.r.t norm $\|\cdot\|$ if $R(y) \geq R(x) + \langle \nabla R(x), y - x \rangle + \frac{\alpha}{2} \|y - x\|^2$

Definition A.4 (Lipschitz continuous gradient smoothness)**.** A differentiable function R is L_R-strongly smooth w.r.t norm $\|\cdot\|$ if $\|\nabla R(x) - \nabla R(y)\|_* \leq L_R \|x - y\|$

Definition A.5 (Bregman Divergence)**.** For a strongly convex regularizer R, $D_R(x, y) = R(x) - R(y) - \langle \nabla R(y), x - y \rangle$ is the Bregman divergence between x and y (not necessarily symmetric)

The following standard result for Bregman divergence will be useful:

Lemma A.1. *[10] For all x, y, z we have the identity $\langle \nabla R(x) - \nabla R(y), x - z \rangle = D_R(x, y) + D_R(z, x) - D_R(z, y)$. Since Bregman divergence is non-negative, a consequence of this identity is that $D_R(z, x) - D_R(z, y) \leq \langle \nabla R(x) - \nabla R(y), z - x \rangle$*

A.2 Expected Regret Bound under Noisy Policy Gradient Estimates and Projection Errors

In this section, we show regret bound for the performance of the sequence of returned programs $\pi_1, \ldots, \pi_T$ of the algorithm. The analysis here is agnostic to the particular implementation of algorithm 2 and algorithm 3.

Let R be a $\alpha-$strongly convex and L_R-smooth functional with respect to norm $\|\cdot\|$ on $\mathcal{H}$. The steps from algorithm 1 can be described as follows.

- Initialize $\pi_0 \in \Pi$. For each iteration t:
  1. Obtain a noisy estimate of the gradient $\widehat{\nabla} J(\pi_{t-1}) \approx \nabla J(\pi_{t-1})$
  2. Update in the $\mathcal{H}$ space: $\nabla R(h_t) = \nabla R(\pi_{t-1}) - \eta \widehat{\nabla} J(\pi_{t-1})$
  3. Obtain approximate projection $\pi_t = \textsc{Project}_\pi^R(h_t) \approx \text{argmin}_{pi \in \Pi} D_R(\pi, h_t)$

This procedure is an approximate functional mirror descent scheme under bandit feedback. We will develop the following result, which is a more detailed version of 4.1 in the main paper.

In the statement below, D is the diameter on Π with respect to defined norm $\|\cdot\|$ (i.e., $D = \sup \|\pi - \pi'\|$). L_J is the Lipschitz constant of the functional J on $\mathcal{H}$. β, σ^2 are the bound on the bias and variance of the gradient estimate at each iteration, respectively. α and $_R$ are the strongly convex and smooth coefficients of the functional regularizer R. Finally, ϵ is the bound on the projection error with respect to the same norm $\|\cdot\|$.

Theorem A.2 (Regret bound of returned policies)**.** *Let $\pi_1, \ldots, \pi_T$ be a sequence of programmatic policies returned by algorithm 1 and π^* be the optimal programmatic policy. We have the expected regret bound:*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) \leq \frac{L_R D^2}{\eta T} + \frac{\epsilon L_R D}{\eta} + \frac{\eta(\sigma^2 + L_J^2)}{\alpha} + \beta D$$

In particular, choosing the learning rate $\eta = \sqrt{\frac{\frac{1}{T}+\epsilon}{\sigma^2}}$, the expected regret is simplified into:

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\sigma\sqrt{\frac{1}{T}+\epsilon} + \beta\right) \tag{4}$$

Proof. At each round t, let $\overline{\nabla}_t = \mathbf{E}[\widehat{\nabla}_t | \pi_t]$ be the conditional expectation of the gradient estimate. We will use the shorthand notation $\nabla_t = \nabla J(\pi_t)$. Denote the upper-bound on the bias of the estimate by β_t, i.e., $\left\|\overline{\nabla}_t - \nabla_t\right\|_* \leq \beta_t$ almost surely. Denote the noise of the gradient estimate by $\xi_t = \overline{\nabla}_t - \widehat{\nabla}_t$, and $\sigma_t^2 = \mathbf{E}\big[\left\|\widehat{\nabla}_t - \overline{\nabla}_t\right\|_*^2\big]$ is the variance of gradient estimate $\widehat{\nabla}_t$.

The projection operator is $\epsilon-$approximate in the sense that $\left\|\pi_t - \textsc{Project}_{\Pi}^{R}(f_t)\right\| = \left\|\widehat{\textsc{Project}}_{\Pi}^{R}(h_t) - \textsc{Project}_{\Pi}^{R}(h_t)\right\| \leq \epsilon$ with some constant ϵ, which reflects the statistical error of the imitation learning procedure. This projection error in general is independent of the choice of function classes Π and $\mathcal{F}$.We will use the shorthand notation $\pi_t^* = \textsc{Project}_{\Pi}^{R}(f_t)$ for the true Bregman projection of h_t onto Π.

Due to convexity of J over the space $\mathcal{H}$ (which includes Π), we have for all $\pi \in \Pi$:

$$J(\pi_t) - J(\pi) \leq \langle \nabla_t, \pi_t - \pi \rangle$$

We proceed to bound the RHS, starting with bounding the inner product where the actual gradient is replaced by the estimated gradient.

$$\langle \widehat{\nabla}_t, \pi_t - \pi \rangle = \frac{1}{\eta_t}\langle \nabla R(\pi_t) - \nabla R(h_{t+1}), \pi_t - \pi \rangle \tag{5}$$

$$= \frac{1}{\eta_t}\big(D_R(\pi, \pi_t) - D_R(\pi, h_{t+1}) + D_R(\pi_t, h_{t+1})\big) \tag{6}$$

$$\leq \frac{1}{\eta_t}\big(D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1}^*) - D_R(\pi_{t+1}^*, h_{t+1}) + D_R(\pi_t, h_{t+1})\big) \tag{7}$$

$$= \frac{1}{\eta_t}\big(\underbrace{D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1})}_{\text{telescoping}} + \underbrace{D_R(\pi, \pi_{t+1}) - D_R(\pi, \pi_{t+1}^*)}_{\text{projection error}} \underbrace{- D_R(\pi_{t+1}^*, h_{t+1}) + D_R(\pi_t, h_{t+1})}_{\text{relative improvement}}\big) \tag{8}$$

Equation (5) is due to the gradient update rule in $\mathcal{F}$ space. Equation (6) is derived from definition of Bregman divergence. Equation (7) is due to the generalized Pythagorean theorem of Bregman projection $D_R(x, y) \geq D_R(x, \textsc{Project}_{\Pi}^{R}(x)) + D_R(\textsc{Project}_{\Pi}^{R}(x), y)$. The RHS of equation (7) are decomposed into three components that will be bounded separately.

Bounding projection error. By lemma (A.1) we have

$$D_R(\pi, \pi_{t+1}) - D_R(\pi, \pi_{t+1}^*) \leq \langle \nabla R(\pi_{t+1}) - \nabla R(\pi_{t+1}^*), \pi - \pi_{t+1} \rangle \tag{9}$$

$$\leq \left\|\nabla R(\pi_{t+1}) - \nabla R(\pi_{t+1}^*)\right\| \left\|\pi - \pi_{t+1}\right\|_* \tag{10}$$

$$\leq L_R \left\|\pi_{t+1} - \pi^*_{t+1}\right\| D \leq \epsilon L_R D \tag{11}$$

Equation (10) is due to Cauchy–Schwarz. Equation (11) is due to Lipschitz smoothness of ∇R and definition of $\epsilon-$approximate projection.

Bounding relative improvement. This follows standard argument from analysis of mirror descent algorithm.

$$D_R(\pi_t, h_{t+1}) - D_R(\pi^*_{t+1}, h_{t+1}) = R(\pi_t) - R(\pi^*_{t+1}) + \langle \nabla R(h_{t+1}), \pi^*_{t+1} - \pi_t \rangle \tag{12}$$

$$\leq \langle \nabla R(\pi_t), \pi_t - \pi^*_{t+1} \rangle - \frac{\alpha}{2} \left\|\pi^*_{t+1} - \pi_t\right\|^2_* + \langle \nabla R(h_{t+1}), \pi^*_{t+1} - \pi_t \rangle \tag{13}$$

$$= -\eta_t \langle \widehat{\nabla}_t, \pi^*_{t+1} - \pi_t \rangle - \frac{\alpha}{2} \left\|\pi^*_{t+1} - \pi_t\right\|^2 \tag{14}$$

$$\leq \frac{\eta_t^2}{2\alpha} \left\|\widehat{\nabla}_t\right\|^2_* \leq \frac{\eta_t^2}{\alpha}(\sigma_t^2 + L_J^2) \tag{15}$$

Equation (13) is from the $\alpha-$strong convexity property of regularizer R. Equation (14) is by definition of the gradient update. Combining the bounds on the three components and taking expectation, we thus have

$$\mathbb{E}\left[\langle \widehat{\nabla}_t, \pi_t - \pi \rangle\right] \leq \frac{1}{\eta_t} \left(D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1}) + \epsilon L_R D + \frac{\eta_t^2}{\alpha}(\sigma_t^2 + L_J^2) \right) \tag{16}$$

Next, the difference between estimated gradient $\widehat{\nabla}_t$ and actual gradient ∇_t factors into the bound via Cauchy-Schwarz:

$$\mathbb{E}\left[\langle \nabla_t - \widehat{\nabla}_t, \pi_t - \pi \rangle\right] \leq \left\|\nabla_t - \mathbb{E}[\widehat{\nabla}_t]\right\|_* \|\pi_t - \pi\| \leq \beta_t D \tag{17}$$

The results can be deduced from equations (16) and (17).

Unbiased gradient estimates. For the case when the gradient estimate is unbiased, assume the variance of the noise of gradient estimates is bounded by σ^2, we have the expected regret bound for all $pi \in \Pi$

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi) \leq \frac{L_R D^2}{\eta T} + \frac{\epsilon L_R D}{\eta} + \frac{\eta(\sigma^2 + L_J^2)}{\alpha} \tag{18}$$

here to clarify, L_R is the smoothness coefficient of regularizer R (i.e., the gradient of R is L_R-Lipschitz, L_J is Lipschitz constant of J, D is the diameter of Π under norm $\|\cdot\|$, σ^2 is the upper-bound on the variance of gradient estimates, and ϵ is the error from the projection procedure (i.e., imitation learning loss).

We can set learning rate $\eta = \sqrt{\frac{\frac{1}{T}+\epsilon}{\sigma^2}}$ to observe that the expected regret is bounded by $O(\sigma\sqrt{\frac{1}{T}+\epsilon})$.

Biased gradient estimates. Assume that the bias of gradient estimate at each round is upper-bounded by $\beta_t \leq \beta$. Similar to before, combining inequalities from (16) and (17), we have

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi) \leq \frac{L_R D^2}{\eta T} + \frac{\epsilon L_R D}{\eta} + \frac{\eta(\sigma^2 + L_J^2)}{\alpha} + \beta D \tag{19}$$

Similar to before, we can set learning rate $\eta = \sqrt{\frac{\frac{1}{T}+\epsilon}{\sigma^2}}$ to observe that on the expected regret is bounded by $O(\sigma\sqrt{\frac{1}{T}+\epsilon} + \beta)$. Compared to the bound on (18), in the biased case, the extra regret incurred per bound is simply a constant, and does not depend on T. □

A.3 Finite-Sample Analysis

In this section, we provide overall finite-sample analysis for PROPEL under some simplifying assumptions. We first consider the case where exact gradient estimate is available, before extending the result to the general case of noisy policy gradient update. Combining the two steps will give us the proof for the following statement (theorem 4.2 in the main paper)

Theorem A.3 (Finite-sample guarantee)**.** *At each iteration, we perform vanilla policy gradient estimate of π (over $\mathcal{H}$) using m trajectories and use DAgger algorithm to collect M roll-outs. Setting*

the learning rate $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}\right)}$*, after* T *rounds of the algorithm, we have that*

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leq O\left(\sigma\sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}\right) + O\left(\sigma\sqrt{\frac{\log(Tk/\delta)}{m}} + \frac{AH\log(Tk/\delta)}{m}\right)$$

holds with probability at least $1-\delta$*, with* H *the task horizon,* A *the cardinality of action space,* σ^2 *the variance of policy gradient estimates, and* k *the dimension* Π*'s parameterization.*

Exact gradient estimate case. Assuming that the policy gradients can be calculated exactly, it is straight-forward to provide high-probability guarantee for the effect of the projection error. We start with the following result, adapted from [45] for the case of projection error bound. In this version of DAgger, we assume that we only collect a single *(state, expert action)* pair from each trajectory roll-out. Result is similar, with tighter bound, when multiple data points are collected along the trajectory.

Lemma A.4 (Projection error bound from imitation learning procedure)**.** *Using DAgger as the imitation learning sub-routine for our* PROJECT *operator in algorithm 3, let* M *be the number of trajectories rolled-out for learning, and* H *be the horizon of the task. With probability at least* $1-\delta$*, we have*

$$D_R(\pi, \pi^*) \leq \widetilde{O}(1/M) + \frac{2\ell_{max}(1+H)}{M} + \sqrt{\frac{2\ell_{max}\log(1/\delta))}{M}}$$

where π *is the result of* PROJECT*,* π^* *is the true Bregman projection of* h *onto* Π*, and* ℓ_{max} *is the maximum value of the imitation learning loss function* $D_R(\cdot,\cdot)$

The bound in lemma A.4 is simpler than previous imitation learning results with cost information ([44, 45]. The reason is that the goal of the PROJECT operator is more modest. Since we only care about the distance between the empirical projection π and the true projection π^*, the loss objective in imitation learning is simplified (i.e., this is only a regret bound), and we can disregard how well policies in Π can imitate the expert h, as well as the performance of $J(\pi)$ relative to the true cost from the environment $J(h)$.

A consequence of this lemma is that for the number of trajectories at each round of imitation learning $M = O(\frac{\log 1/\delta}{\epsilon^2}) + O(\frac{H}{\epsilon})$, we have $D_R(\pi_t, \pi_t^*) \leq \epsilon$ with probability at least $1-\delta$. Applying union bound across T rounds of learning, we obtain the following guarantee (under no gradient estimation error)

Proposition A.5 (Finite-sample Projection Error Bound)**.** *To simplify the presentation of the result, we consider* L_R, D, L, α *to be known constants. Using DAgger algorithm to collect* $M = O(\frac{\log T/\delta}{\epsilon^2}) + O(\frac{H}{\epsilon})$ *roll-outs at each iteration, we have the following regret guarantee after* T *rounds of our main algorithm:*

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leq O\left(\frac{1}{\eta T} + \frac{\epsilon}{\eta} + \eta\right)$$

with probability at least $1-\delta$*. Consequently, setting* $\eta = \sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}$*, we have that*

$$\frac{1}{T}\sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leq O\left(\sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}}\right)$$

with probability at least $1-\delta$

Note that the dependence on the time horizon of the task is sub-linear. This is different from standard imitation learning regret bounds, which are often at least linear in the task horizon. The main reason is that our comparison benchmark π^* does live in the space Π, whereas for DAgger, the expert policy may not reside in the same space.

Noisy gradient estimate case. We now turn to the issue of estimating the gradient of $\nabla J(\pi)$. We make the following simplifying assumption about the gradient estimation:

- The π is parameterized by vector $\theta \in \mathbb{R}^k$ (such as a neural network). The parameterization is differentiable with respect to θ (Alternatively, we can view Π as a differentiable subspace of $\mathcal{F}$, in which case we have $\mathcal{H} = \mathcal{F}$)
- At each UPDATE loop, the policy is rolled out m times to collect the data, each trajectory has horizon length H
- For each visited state $s \sim d_h$, the policy takes a uniformly random action a. The action space is finite with cardinality A.
- The gradient ∇h_θ is bounded by B

The gradient estimate is performed consistent with a generic policy gradient scheme, i.e.,

$$\widehat{\nabla} J(\theta) = \frac{A}{m} \sum_{i=1}^{H} \sum_{j=1}^{m} \nabla \pi_\theta(a_i^j | s_i^j, \theta) \widehat{Q}_i^j$$

where $\widehat{Q}_i^j$ is the estimated cost-to-go [55].

Taking uniform random exploratory actions ensures that the samples are i.i.d. We can thus apply Bernstein's inequality to obtain the bound between estimated gradient and the true gradient. Indeed, with probability at least $1 - \delta$, we have that the following bound on the bias component-wise:

$$\left\| \widehat{\nabla} J(\theta) - \nabla J(\theta) \right\|_\infty \leq \beta \text{ when } m \geq \frac{(2\sigma^2 + 2AHB\frac{\beta}{3}) \log \frac{k}{\delta}}{\beta^2}$$

which leads to similar bound with respect to $\|\cdot\|_*$ (here we leverage the equivalence of norms in finite dimensional setting):

$$\left\| \nabla_t - \widehat{\nabla}_t \right\|_* \leq \beta \text{ when } m = O\left(\frac{(\sigma^2 + AHB\beta) \log \frac{k}{\delta}}{\beta^2} \right)$$

Applying union bound of this result over T rounds of learning, and combining with the result from proposition (A.5), we have the following finite-sample guarantee in the simplifying policy gradient update. This is also the more detailed statement of theorem 4.2 in the main paper.

Proposition A.6 (Finite-sample Guarantee under Noisy Gradient Updates and Projection Error)**.** *At each iteration, we perform policy gradient estimate using $m = O(\frac{(\sigma^2 + AHB\beta) \log \frac{Tk}{\delta}}{\beta^2})$ trajectories and use DAgger algorithm to collect $M = O(\frac{\log T/\delta}{\epsilon^2}) + O(\frac{H}{\epsilon})$ roll-outs. Setting the learning rate $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}\right)}$, after T rounds of the algorithm, we have that*

$$\frac{1}{T} \sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leq O\left(\sigma \sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}} \right) + \beta$$

with probability at least $1 - \delta$.

Consequently, we also have the following regret bound:

$$\frac{1}{T} \sum_{t=1}^{T} J(\pi_t) - J(\pi^*) \leq O\left(\sigma \sqrt{\frac{1}{T} + \frac{H}{M} + \sqrt{\frac{\log(T/\delta)}{M}}} \right) + O\left(\sigma \sqrt{\frac{\log(Tk/\delta)}{m}} + \frac{AH \log(Tk/\delta)}{m} \right)$$

holds with probability at least $1 - \delta$, where again H is the task horizon, A is the cardinality of action space, and k is the dimension of function class Π's parameterization.

Proof. (For both proposition (A.6) and (A.5)). The results follow by taking the inequality from equation (19), and by solving for ϵ and β explicitly in terms of relevant quantities. Based on the specification of M and m, we obtain the necessary precision for each round of learning in terms of number of trajectories:

$$\beta = O(\sigma \frac{\log(k/\delta)}{m} + \frac{AHB \log(k/\delta)}{m})$$
$$\epsilon = O(\frac{H}{M} + \sqrt{\frac{\log(1/\delta)}{M}})$$

Setting the learning rate $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \epsilon\right)}$ and rearranging the inequalities lead to the desired bounds. □

The regret bound depends on the variance σ^2 of the policy gradient estimates. It is well-known that vanilla policy gradient updates suffer from high variance. We instead use functional regularization technique, based on CORE-RL, in the practical implementation of our algorithm. The CORE-RL subroutine has been demonstrated to reduce the variance in policy gradient updates [19].

A.4 Defining a consistent approximation of $\nabla_{\mathcal{H}} J(\pi)$ - Proof of Proposition 4.3

We are using the notion of Fréchet derivative to define gradient of differentiable functional. Note that while Gateaux derivative can also be utilized, Fréchet derivative ensures continuity of the gradient operator that would be useful for our analysis.

Definition A.6 (Fréchet gradient)**.** A bounded linear operator $\nabla : \mathcal{H} \mapsto \mathcal{H}$ is called Fréchet functional gradient of J at $h \in \mathcal{H}$ if $\lim_{\|g\| \to 0} \frac{J(h+g) - J(h) - \langle \nabla J(h), g \rangle}{\|g\|} = 0$

We make the following assumption about $\mathcal{H}$ and $\mathcal{F}$. One interpretation of this assumption is that the space of policies Π and $\mathcal{F}$ that we consider have the property that a programmatic policy $\pi \in \Pi$ can be well-approximated by a large space of neural policies $f \in \mathcal{F}$.

Assumption 1. *J is Fréchet differentiable on $\mathcal{H}$. J is also differentiable on the restricted subspace $\mathcal{F}$. And $\mathcal{F}$ is dense in $\mathcal{H}$ (i.e., the closure $\overline{\mathcal{F}} = \mathcal{H}$)*

It is then clear that $\forall\ f \in \mathcal{F}$ the Fréchet gradient $\nabla_{\mathcal{F}} J(f)$, restricted to the subspace $\mathcal{F}$ is equal to the gradient of f in the ambient space $\mathcal{H}$ (since Fréchet gradient is unique). In general, given $\pi \in \Pi$ and $f \in \mathcal{F}$, $\pi + f$ is not necessarily in $\mathcal{F}$. However, the restricted gradient on subspace $\mathcal{F}$ of $J(\pi + f)$ can be defined asymptotically.

Proposition A.7. *Fixing a policy $\pi \in \Pi$, define a sequence of policies $f_k \in \mathcal{F}$, $k = 1, 2, \ldots$ that converges to π: $\lim_{k\to\infty} \|f_k - g\| = 0$, we then have $\lim_{k\to\infty} \|\nabla_{\mathcal{F}} J(f_k) - \nabla_{\mathcal{H}} J(\pi)\|_* = 0$*

Proof. Since Fréchet derivative is a continuous linear operator, we have $\lim_{k\to\infty} \|\nabla_{\mathcal{H}} J(f_k) - \nabla_{\mathcal{H}} J(\pi)\|_* = 0$. By the reasoning above, for $f \in \mathcal{F}$, the gradient $\nabla_{\mathcal{F}} J(f)$ defined via restriction to the space $\mathcal{F}$ does not change compared to $\nabla_{\mathcal{H}} J(f)$, the gradient defined over the ambient space $\mathcal{H}$. Thus we also have $\lim_{k\to\infty} \|\nabla_{\mathcal{F}} J(f_k) - \nabla_{\mathcal{H}} J(\pi)\|_* = 0$. By the same argument, we also have that for any given $\pi \in \Pi$ and $f \in \mathcal{F}$, even if $\pi + f \notin \mathcal{F}$, the gradient $\nabla_{\mathcal{F}} J(\pi + f)$ with respect to the $\mathcal{F}$ can be approximated similarly. □

Note that we are not assuming $J(\pi)$ to be differentiable when restricting to the policy subspace Π.

A.5 Theoretical motivation for Algorithm 2 - Proof of Proposition 4.4 and 4.5

We consider the case where Π is not differentiable by parameterization. Note that this does not preclude $J(\pi)$ for $\pi \in \Pi$ to be differentiable in the non-parametric function space. Two complications arise compared to our previous approximate mirror descent procedure. First, for each $\pi \in \Pi$, estimating the gradient $\nabla J(\pi)$ (which may not exist under certain parameterization, per section 4.3) can become much more difficult. Second, the update rule $\nabla R(\pi) - \nabla_{\mathcal{F}} J(\pi)$ may not be in the dual space of $\mathcal{F}$, as in the simple case where $\Pi \subset \mathcal{F}$, thus making direct gradient update in the $\mathcal{F}$ space inappropriate.

Assumption 2. *J is convex in $\mathcal{H}$.*

By convexity of J in $\mathcal{H}$, sub-gradients $\partial J(h)$ exists for all $h \in \mathcal{H}$. In particular, $\partial J(\pi)$ exists for all $\pi \in \Pi$. Note that $\partial J(\pi)$ reflects sub-gradient of π with respect to the ambient policy space $\mathcal{H}$.

We will make use of the following equivalent perspective to mirror descent[10], which consists of two-step process for each iteration t

1. Solve for $h_{t+1} = \operatorname{argmin}_{h \in \mathcal{H}} \eta \langle \partial J(\pi_t), h \rangle + D_R(h, \pi_t)$
2. Solve for $\pi_{t+1} = \operatorname{argmin}_{\pi \in \Pi} D_R(\pi, h_{t+1})$

We will show how this version of the algorithm motivates our main algorithm. Consider step 1 of the main loop of PROPEL, where given a fixed $\pi \in \Pi$, the optimization problem within $\mathcal{H}$ is

$$(\text{OBJECTIVE\_1}) = \min_{h \in \mathcal{H}} \eta \langle \partial J(\pi), h \rangle + D_R(h, \pi) \tag{20}$$

Due to convexity of $\mathcal{H}$ and the objective, problem (OBJECTIVE_1) is equivalent to:

$$(\text{OBJECTIVE\_1}) = \min \langle \partial J(\pi), h \rangle \tag{21}$$

$$\text{s.t. } D_R(h, \pi) \leq \tau \tag{22}$$

where τ depends on η. Since π is fixed, this optimization problem can be relaxed by choosing $\lambda \in [0, 1]$, and a set of candidate policies $h = \pi + \lambda f$, for all $f \in \mathcal{F}$, such that $D_R(h, \pi) \leq \tau$ is satisfied (Selection of λ is possible with bounded spaces). Since this constraint set is potentially a restricted set compared to the space of policies satisfying inequality (22), the optimization problem (20) is relaxed into:

$$(\text{OBJECTIVE\_1}) \leq (\text{OBJECTIVE\_2}) = \min_{f \in \mathcal{F}} \langle \partial J(\pi), \pi + \lambda f \rangle \tag{23}$$

Due to convexity property of J, we have

$$\langle \partial J(\pi), \lambda f \rangle = \langle \partial J(\pi), \pi + \lambda f - \pi) \rangle \leq J(\pi + \lambda f) - J(\pi) \tag{24}$$

The original problem OBJECTIVE_1 is thus upper bounded by:

$$\min_{h \in \mathcal{H}} \eta \langle \partial J(\pi), h) \rangle + D_R(h, \pi) \leq \min_{f \in \mathcal{F}} J\big(\pi + \lambda f\big) - J(\pi) + \langle \partial J(\pi), \pi \rangle$$

Thus, a relaxed version of original optimization problem OBJECTIVE_1 can be obtained by minizimiing $J(\pi + \lambda f)$ over $f \in \mathcal{F}$ (note that π is fixed). This naturally motivates using functional regularization technique, such as CORE-RL algorithm [19], to update the parameters of differentiable function f via policy gradient descent update:

$$f' = f - \eta \lambda \nabla_{\mathcal{F}} \lambda J(\pi + \lambda f)$$

where the gradient of J is taken with respect to the parameters of f (neural networks). This is exactly the update step in algorithm 2 (also similar to iterative updte of CORE-RL algorithm), where the neural network policy is regularized by a prior controller π.

Statement and Proof of Proposition 4.5

Proposition A.8 (Regret bound for the relaxed optimization objective)**.** *Assuming $J(h)$ is L-strongly smooth over $\mathcal{H}$, i.e., $\nabla_{\mathcal{H}} J(h)$ is L-Lipschitz continuous, approximating* $\text{UPDATE}_{\mathcal{H}}$ *by* UPDATE_F *per Alg. 2 leads to the expected regret bound:* $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(\pi_t)\right] - J(\pi^*) = O\left(\lambda\sigma\sqrt{\frac{1}{T} + \epsilon} + \lambda^2 L^2\right)$

Proof. Instead of focusing on the bias of the gradient estimate $\nabla_{\mathcal{H}} J(\pi)$, we will shift our focus on the alternative proximal formulation of mirror descent, under optimization and projection errors. In particular, at each iteration t, let $h^*_{t+1} = \text{argmin}_{h \in \mathcal{H}} \eta \langle \nabla J(\pi_t), h \rangle + D_R(h, \pi_t)$ and let the optimization error be defined as β_t where $\nabla R(h_{t+1}) = \nabla R(h^*_{t+1}) + \beta_t$. Note here that this is different from (but related to) the notion of bias from gradient estimate of $\nabla J(\pi)$ used in theorem 4.1 and theorem A.2. The projection error from imitation learning procedure is defined similarly to theorem 4.1: $\pi^*_{t+1} = \text{argmin}_{\pi \in \Pi} D_R(\pi, h_{t+1})$ is the true projection, and $\left\|\pi_{t+1} - \pi^*_{t+1}\right\| \leq \epsilon$.

We start with similar bounding steps to the proof of theorem 4.1:

$$\begin{aligned}
\langle \nabla J(\pi_t), \pi_t - \pi \rangle &= \frac{1}{\eta} \langle \nabla R(h^*_{t+1}) - \nabla R(\pi_t), \pi_t - \pi \rangle \\
&= \frac{1}{\eta} \left(\langle \nabla R(h_{t+1}) - \nabla R(\pi_t), \pi_t - \pi \rangle - \langle \beta_t, \pi_t - \pi \rangle \right) \\
&= \underbrace{\frac{1}{\eta} \left(D_R(\pi, \pi_t) - D_R(\pi, h_{t+1}) + D_R(\pi_t, h_{t+1}) \right)}_{\text{component_1}} + \underbrace{\frac{1}{\eta} \langle \beta_t, \pi_t - \pi \rangle}_{\text{component_2}}
\end{aligned} \tag{25}$$

As seen from the proof of theorem A.2, component_1 can be upperbounded by: $\frac{1}{\eta}\big(\underbrace{D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1})}_{\text{telescoping}} + \underbrace{D_R(\pi, \pi_{t+1}) - D_R(\pi, \pi^*_{t+1})}_{\text{projection error}} \underbrace{- D_R(\pi^*_{t+1}, h_{t+1}) + D_R(\pi_t, h_{t+1})}_{\text{relative improvement}} \big)$

The bound on projection error is identical to theorem A.2:

$$D_R(\pi, \pi_t) - D_R(\pi, \pi^*_{t+1}) \leq \epsilon L_R D \tag{26}$$

The bound on relative improvement is slightly different:

$$
\begin{aligned}
&D_R(\pi_t, h_{t+1}) - D_R(\pi^*_{t+1}, h_{t+1}) = R(\pi_t) - R(\pi^*_{t+1}) + \langle \nabla R(h_{t+1}), \pi^*_{t+1} - \pi_t \rangle \\
&= R(\pi_t) - R(\pi^*_{t+1} + \langle \nabla R(h^*_{t+1}), \pi^*_{t+1} - \pi_t \rangle) + \langle \beta_t, \pi^*_{t+1} - \pi_t \rangle \\
&\leq \langle \nabla R(\pi_t), \pi_t - \pi^*_{t+1} \rangle - \frac{\alpha}{2} \left\| \pi^*_{t+1} - \pi_t \right\|^2 + \langle \nabla R(h^*_{t+1}), \pi^*_{t+1} - \pi_t \rangle + \langle \beta_t, \pi^*_{t+1} - \pi_t \rangle \\
&= -\eta \langle \nabla J_{\mathcal{H}}(\pi_t), \pi^*_{t+1} - \pi_t \rangle - \frac{\alpha}{2} \left\| \pi^*_{t+1} - \pi_t \right\|^2 + \langle \beta_t, \pi^*_{t+1} - \pi_t \rangle \qquad (27) \\
&\leq \frac{\eta^2}{2\alpha} \left\| \nabla_{\mathcal{H}} J(\pi_t) \right\|_*^2 + \langle \beta_t, \pi^*_{t+1} - \pi_t \rangle \\
&\leq \frac{\eta^2}{2\alpha} L_J^2 + \langle \beta_t, \pi^*_{t+1} - \pi_t \rangle \qquad (28)
\end{aligned}
$$

Note here that the gradient $\nabla_{\mathcal{H}} J(\pi_t)$ is not the result of estimation. Combining equations (25), (26), (27), (28), we have:

$$
\langle \nabla J(\pi_t), \pi_t - \pi \rangle \leq \frac{1}{\eta} \big(D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1}) + \epsilon L_R D + \frac{\eta^2}{2\alpha} L_J^2 + \langle \beta_t, \pi^*_{t+1} - \pi \rangle \big) \qquad (29)
$$

Next, we want to bound β_t. Choose regularizer R to be $\frac{1}{2} \left\| \cdot \right\|^2$ (consistent with the pseudocode in algorithm 2). We have that:

$$
h^*_{t+1} = \operatorname*{argmin}_{h \in \mathcal{H}} \eta \langle \nabla J(\pi_t), h \rangle + \frac{1}{2} \left\| h - \pi_t \right\|^2
$$

which is equivalent to:

$$
h^*_{t+1} = \pi_t + \operatorname*{argmin}_{f \in \mathcal{F}} \eta \langle \nabla J(\pi_t), f \rangle + \frac{1}{2} \left\| f \right\|^2
$$

Let $f^*_{t+1} = \operatorname{argmin}_{f \in \mathcal{F}} \eta \langle \nabla J(\pi_t), f \rangle + \frac{1}{2} \left\| f \right\|^2$. Taking the gradient over f, we can see that $f^*_{t+1} = -\eta \nabla J(\pi_t)$. Let f_{t+1} be the minimizer of $\min_{f \in \mathcal{F}} J(\pi_t + \lambda f)$. We then have $h^*_{t+1} = \pi_t + f^*_{t+1}$ and $h_{t+1} = \pi + \lambda f_{t+1}$. Thus $\beta_t = h_{t+1} - h^*_{t+1} = f_{t+1} - f^*_{t+1}$.

On one hand, we have

$$
J(\pi_t + \lambda f_{t+1}) \leq J(\pi_t + \omega f^*_{t+1}) \leq J(\pi_t) + \langle \nabla J(\pi_t), \omega f^*_{t+1} \rangle + \frac{L}{2} \left\| \omega f^*_{t+1} \right\|^2
$$

due to optimality of f_{t+1} and strong smoothness property of J. On the other hand, since J is convex, we also have the first-order condition:

$$
J(\pi_t + \lambda f_{t+1}) \geq J(\pi_t) + \langle \nabla J(\pi_t), \lambda f_{t+1} \rangle
$$

Combine with the inequality above, and subtract $J(\pi_t)$ from both sides, and using the relationship $f^*_{t+1} = -\eta \nabla J(\pi_t)$, we have that:

$$
\langle -\frac{1}{\eta} f^*_{t+1}, \lambda f_{t+1} \rangle \leq \langle -\frac{1}{\eta} f^*_{t+1}, \omega f^*_{t+1} \rangle + \frac{L\omega^2}{2} \left\| f^*_{t+1} \right\|^2
$$

Since this is true $\forall \omega$, rearrange and choose ω such that $\frac{\omega}{\eta} - \frac{L\omega^2}{2} = -\frac{\lambda}{2\eta}$, namely $\omega = \frac{1 - \sqrt{1 - \lambda\eta L}}{L\eta}$, and complete the square, we can establish the bound that:

$$
\left\| f_{t+1} - f^*_{t+1} \right\| \leq \eta (\lambda L)^2 B \qquad (30)
$$

for B the upperbound on $\|f_{t+1}\|$. We thus have $\|\beta_t\| = O(\eta(\lambda L)^2)$. Plugging the result from equation 30 into RHS of equation 29, we have:

$$
\langle \nabla J(\pi_t), \pi_t - \pi \rangle \leq \frac{1}{\eta} \big(D_R(\pi, \pi_t) - D_R(\pi, \pi_{t+1}) + \epsilon L_R D + \frac{\eta^2}{2\alpha} L_J^2 \big) + \big(\eta (\lambda L)^2 B \big) \qquad (31)
$$

Since J is convex in $\mathcal{H}$, we have $J(\pi_t) - J(\pi) \leq \langle \nabla J(\pi_t), \pi_t - \pi \rangle$. Similar to theorem 4.1, setting $\eta = \sqrt{\frac{1}{\lambda^2 \sigma^2} (\frac{1}{T} + \epsilon)}$ and taking expectation on both sides, we have:

$$
\mathbb{E} \left[\frac{1}{T} \sum_{t=1}^{T} J(\pi_t) \right] - J(\pi^*) = O\big(\lambda \sigma \sqrt{\frac{1}{T} + \epsilon} + \lambda^2 L^2 \big) \qquad (32)
$$

Note that unlike regret bound from theorem 4.1 under general bias, variance of gradient estimate and projection error, σ^2 here explicitly refers to the bound on neural-network based policy gradient

variance. The variance reduction of $\lambda\sigma$, at the expense of some bias, was also similarly noted in a recent functional regularization technique for policy gradient [19]. □

B Additional Experimental Results and Details

B.1 TORCS

We generate controllers for cars in *The Open Racing Car Simulator* (TORCS) [59]. In its full generality TORCS provides a rich environment with input from up to 89 sensors, and optionally the 3D graphic from a chosen camera angle in the race. The controllers have to decide the values of 5 parameters during game play, which correspond to the acceleration, brake, clutch, gear and steering of the car.

Apart from the immediate challenge of driving the car on the track, controllers also have to make race-level strategy decisions, like making pit-stops for fuel. A lower level of complexity is provided in the Practice Mode setting of TORCS. In this mode all race-level strategies are removed. Currently, so far as we know, state-of-the-art DRL models are capable of racing only in Practice Mode, and this is also the environment that we use. Here we consider the input from 29 sensors, and decide values for the acceleration, steering, and braking actions.

We chose a suite of tracks that provide varying levels of difficulty for the learning algorithms. In particular, for the tracks Ruudskogen and Alpine-2, the DDPG agent is unable to reliably learn a policy that would complete a lap. We perform the experiments with twenty-five random seeds and report the median lap time over these twenty-five trials. However we note that the TORCS simulator is not deterministic even for a fixed random seed. Since we model the environment as a Markov Decision Process, this non-determinism is consistent with our problem statement.

For our Deep Reinforcement Learning agents we used standard open source implementations (with pre-tuned hyper-parameters) for the relevant domain.

All experiments were conducted on standard workstation with a 2.5 GHz Intel Core i7 CPU and a GTX 1080 Ti GPU card.

The code for the TORCS experiments can be found at: https://bitbucket.org/averma8053/propel

In Table 3 we show the lap time performance and crash ratios of PROPEL agents initialized with neural policies obtained via DDPG. As discussed in Section 5, DDPG often exhibits high variance across trials and this adversely affects the performance of the PROPEL agents when they are initialized via DDPG. In Table 4 we show generalization results for the PROPELTREE agent. As noted in Section 5, the generalization results for PROPELTREE are in between those of DDPG and PROPELPROG.

Verified Smoothness Property. For the program given in Figure 2 we proved using symbolic verification techniques, that $\forall k,\ \sum_{i=k}^{k+5} \|\mathbf{peek}(s[\texttt{RPM}], i+1) - \mathbf{peek}(s[\texttt{RPM}], i)\| < 0.003 \implies \|\mathbf{peek}(a[\texttt{Accel}], k+1) - \mathbf{peek}(a[\texttt{Accel}], k)\| < 0.63$. Here the function $\mathbf{peek}(., i)$ takes in a history/sequence of sensor or action values and returns the value at position i, . Intuitively, the above logical implication means that if the sum of the consecutive differences of the last six RPM sensor values is less than 0.003, then the acceleration actions calculated at the last and penultimate step will not differ by more than 0.63.

Table 3: *Performance results in* TORCS *of* PROPEL *agents initialized with neural policies obtained via* DDPG*, over 25 random seeds. Each entry is formatted as Lap-time / Crash-ratio, reporting median lap time in seconds over all the seeds (lower is better) and ratio of seeds that result in crashes (lower is better). A lap time of* CR *indicates the agent crashed and could not complete a lap for more than half the seeds.*

LENGTH	G-TRACK 3186M	E-ROAD 3260M	AALBORG 2588M	RUUDSKOGEN 3274M	ALPINE-2 3774M
PROPELPROG-DDPG	97.76/.12	108.06/.08	140.48/.48	CR / 0.68	CR / 0.92
PROPELTREE-DDPG	78.47/0.16	85.46/.04	CR / 0.56	CR / 0.68	CR / 0.92

Table 4: *Generalization results in* TORCS *for* PROPELTREE*, where rows are training and columns are testing tracks. Each entry is formatted as* PROPELPROG */ DDPG, and the number reported is the median lap time in seconds over all the seeds (lower is better).* CR *indicates the agent crashed and could not complete a lap for more than half the seeds.*

	G-TRACK	E-ROAD	AALBORG	RUUDSKOGEN	ALPINE-2
G-TRACK	-	95	CR	CR	CR
E-ROAD	84	-	CR	CR	CR
AALBORG	111	CR	-	CR	CR
RUUDSKOGEN	154	CR	CR	-	CR
ALPINE-2	CR	276	CR	CR	-

Table 5: Performance results in Classic Control problems. Higher scores are better.

	MOUNTAINCAR	PENDULUM
PRIOR	00.59 ± 0.00	-875.53 ± 0.00
DDPG	97.16 ± 3.21	-132.70 ± 6.44
TRPO	93.03 ± 1.86	-131.54 ± 4.56
NDPS	66.98 ± 3.11	-435.71 ± 4.83
VIPER	64.86 ± 3.28	-394.11 ± 4.97
PROPELPROG	95.63 ± 1.02	-187.71 ± 2.35
PROPELTREE	96.56 ± 2.81	-139.09 ± 3.31

B.2 Classic Control

We present results from two classic control problems, Mountain-Car (with continuous actions) and Pendulum, in Table 5. We use the OpenAI Gym implementations of these environments. More information about these environments can be found at the links: MountainCar and Pendulum.

In Mountain-Car the goal is to drive an under-powered car up the side of a mountain in as few time-steps as possible. In Pendulum, the goal is to swing a pendulum up so that it stays upright. In both the environments an episode terminates after a maximum of 200 time-steps.

In Table 5 we report the mean and standard deviation, over twenty-five random seeds, of the average scores over 100 episodes for the listed agents and environments. In Figure 6 and Figure 7 we show the improvements made over the prior by the PROPELPROG agent in MountainCar and Pendulum respectively, with each iteration of the PROPEL algorithm.

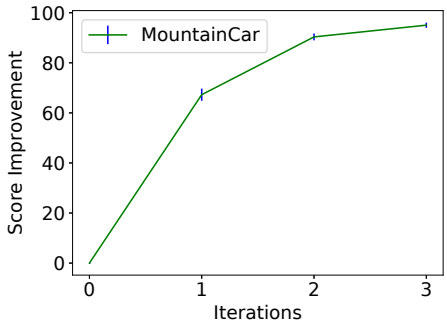

Figure 6: Score improvements in the MountainCar environment over iterations of PROPELPROG.

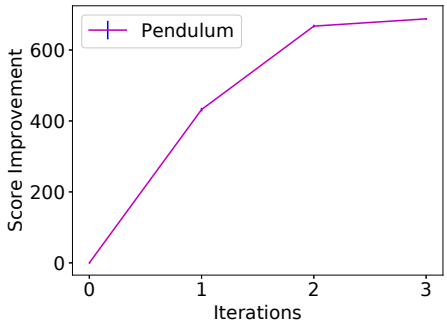

Figure 7: Score improvements in the Pendulum environment over iterations of PROPELPROG.